The Journal of Logic and Algebraic Programming 78 (2009) 730–748

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

A barred operational semantics for a subset of WS-CDL with time restrictions☆

Valentín Valero, Gregorio Díaz*, María Emilia Cambronero, Hermenegilda Macià

Escuela Superior de Ingeniería Informática de Albacete, Universidad de Castilla-La Mancha, Campus Universitario s/n, 02071. Albacete, Spain

ARTICLE INFO

Article history:
Received 8 April 2008
Accepted 30 July 2009
Available online 6 August 2009

Keywords:
Web Services
Web Service composition
Timed interactions
Choreography
Operational semantics

ABSTRACT

Web Services composition provides a way to obtain value-added services by combining several Web Services. WS-CDL (Web Services Choreography Description Language) is a W3C candidate recommendation for the description of peer-to-peer collaborations for the participants in a Web Services composition. However, the semantics of WS-CDL is provided in a textual way, and hence a complete rigorous semantics is lacking. In this paper we focus our attention on the WS-CDL elements related to concurrency, as well as on the collaborations that have timing restrictions associated. We then provide an operational semantics for a relevant subset of WS-CDL, paying special attention to timed collaborations. This operational semantics is based on barred terms, which allow us to capture the current state of the choreography throughout its execution.

1. Introduction

A Web Service can be defined [1] as a self-describing, self-contained modular application that can be published, located and invoked over a network, usually the Internet. Web Services are becoming more and more important as a platform for B2B integration. Web Service composition has appeared as a natural and elegant way to provide new value-added services as a combination of several established Web Services. Services provided by different suppliers can act together to provide another service; in fact, they can be written in different languages and can be executed on different platforms.

Internet and Web technologies are thus a new way of doing business, more cheaply and efficiently, as enterprises can provide new and dynamic services faster by the composition of Web Services. But B2B e-commerce is still emerging, and new software technologies are being required to support their development. Specifically, there is a need for effective and efficient means to abstract, compose, analyze, and evolve Web Services within an appropriate time-frame [12].

The current technology is based on the Service Oriented Architecture (SOA) stack, proposed by the World Wide Web Consortium, W3C [21], in which the top layers are the orchestration and choreography levels. On the orchestration level the execution logic of Web Services based applications is described by defining their control flows (such as conditional, sequential, parallel and exceptional execution) and prescribing the rules for consistently managing their non-observable data. Thus, the orchestration refers to automated execution of a workflow, using an execution language such as WS-BPEL [2].

☆ Supported by the Spanish government (cofinanced by FEDER founds) with the Project TIN2006-15578-C02-02, and the JCCLM regional Project PAC06-0008-6995.

* Corresponding author.
E-mail addresses: Valentin.Valero@uclm.es (V. Valero), Gregorio.Diaz@uclm.es (G. Díaz), MEmilia.Cambronero@uclm.es (M.E. Cambronero), Hermenegilda.Macia@uclm.es (H. Macià).

1567-8326/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2009.07.001

At the top of the SOA stack we have the choreography layer, which refers to a description of coordinated interactions between two or more parties. A Web Services choreography describes peer-to-peer collaborations of the Web Services choreography participants by defining, from a global viewpoint, their common and complementary observable behavior, where ordered message exchanges result in accomplishing a common business goal. For that purpose the W3C has proposed WS-CDL (Web Services Choreography Description Language) [22] as a candidate recommendation for the description of choreographies. This is an XML-based description language, which allows us to describe the collaborations between the different parties involved in a composite Web Service. In a WS-CDL document these parties are identified and the interactions between them are described, indicating the order in which they occur and the messages exchanged.

However, the semantics of WS-CDL on the official site [22] is defined simply in a textual manner, so that the language definition lacks formalization. One of the main goals of this paper, then, is the formalization of the WS-CDL semantics, more specifically, the definition of an operational semantics and a bisimulation relation for a relevant subset of WS-CDL, which includes the main activity constructions of the language (basic and structured), integer variables, exception blocks, and also the timing restrictions introduced by the WS-CDL interactions. This operational semantics is a barred semantics, similar to that used to define the operational semantics of the Petri Box Calculus PBC [4], in the sense that we use terms annotated with both overbars and underbars in order to set up the current state of the system.

This semantics covers the main aspects of WS-CDL, being based on its main structural elements, namely, choreographies, activities and exception blocks. It also takes into account variables, with their values ranging over an integer domain, and the exceptions that could arise when using unassigned variables. Furthermore, another important aspect covered by WS-CDL is that of timing restrictions in the inter-party collaborations, we may have, for instance, time-outs associated with interactions, or interactions that must be executed after a certain delay or at a certain instant. In this semantics, special attention is paid to this aspect, so that the abstract language that we use to capture the syntax of the considered subset of WS-CDL is actually a timed language. The time model we consider is discrete, there being a special transition that captures the passage of one time unit. The alternative to discrete time models are continuous time models, which are normally preferred, as they use a real time scale, so that, in principle, the execution of actions is not restricted to discrete points in time. However, as Baeten and Middelburg mention in [3] measuring time on a discrete time scale must not be seen as enforcing the execution of actions in specific time instants, but the division of time into slices and the timing of actions is carried out with respect to the time slices in which they are performed. Computers, in fact, measure time by means of discrete clocks, and if they are used to control a physical system, the state of the physical system is sampled and adjusted at discrete points in time.

2. Related work

The developers of WS-CDL claim that its design has been based on a formal language, the π-calculus [17], and that therefore WS-CDL is a particularly well-suited language for describing concurrent processes and dynamic interconnection scenarios. This relationship has been studied in [9,10], where the authors compare a formalized version of WS-CDL, called global calculus, with the π-calculus. The global calculus is inspired by WS-CDL, thus it contains operators for some of the different activities of WS-CDL: inactions, communications, assignments, conditionals, choices, parallels, hiding and recursion. In contrast with this formalism, in this paper we present a model with a closer syntax to that of WS-CDL, paying special attention to timed aspects of WS-CDL. The importance of timing issues in e-business has been recognized in [15], where a timed calculus inspired in the π-calculus is defined. However, in that work time is only included in a timed extension of the workunit construction. Furthermore, in that work a simulation result is presented, but not a bisimulation relation.

Another related work is COWS (Calculus for Orchestration of Web Services) [13], which is a process calculus for Service Oriented Computing whose design has been influenced by WS-BPEL, although it is not specifically tied to any current Web Services description language. COWS permits the modeling of different but typical aspects of (web) services technologies, such as multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions. COWS, then, focuses on service orchestration rather than on service choreography, and omits any consideration of service timing aspects. In [14] a π-calculus based semantics for WS-BPEL is presented, where special attention is given to event, fault, and compensation handlers behavior.

The relationship between choreography and orchestration has been studied in some depth in [7,8], where conformability relations are introduced.

There are some other works that use algebraic models in the area of Web Services composition: a relation between the traditional process algebra operators and the main aspects of Service Oriented Computing is presented in [5], where the time-out mechanism is mentioned as a key element in the Web Services area. The authors analyse three specific peculiarities of service oriented computing from the orchestration viewpoint: loose coupling, communication latency and open-endedness.

In [19] the authors have defined a process algebra to derive the interactive behavior of a business process starting from a WS-BPEL specification. In [6] a translation of WSCI (Web Service Choreography Interface) to CCS [16] is defined, showing the benefits of such a translation. WSCI [23] is also an XML-based language that describes the global behavior of the participants of a composite Web Service from a choreographic viewpoint. Although there are some similarities between WSCI and WS-CDL, the later has gained more attention because WS-CDL is more complete, it deals with variables, with complex interactions, and with *workunits*, a more sophisticated structure that the *activity sets* of WSCI to join activities that should be executed under certain conditions.

In [25] a mapping from WS-CDL and WS-BPEL into CSP is presented, providing a formal approach to verifying the behavior of collaborating Web services. However, this translation is only provided for a very limited subset of WS-CDL, it is presented in a schematic way, and neither variables nor timed aspects are considered. In [24] the authors have also made a translation of a subset of WS-CDL into a formal model, in this case a small language (CDL), for which they provide an operational semantics. This work has been recently extended [18] by including a projection of the choreography level onto the orchestration level, the dominant role concept being introduced which is used in the implementation of any choice or interaction structure of the choreography. In both works, however, neither data or timing aspects are considered. Finally, we have also looked closely at translations of WS-CDL with time and priorities to other formalisms, such as Petri Nets [20] or timed automata [11].

We have structured the paper as follows: in Section 3 we present a brief description of the main elements of WS-CDL. The operational semantics of a relevant subset WS-CDL is defined in Section 4, and a case study is introduced in Section 5. Finally, in Section 6 the conclusions and some indications about our future work are formulated.

3. WS-CDL

The Web Services Choreography specification offers a precise description of collaborations between the parties involved in a choreography. WS-CDL specifications are contracts containing "global" definitions of the common ordering conditions and constraints under which messages are exchanged. The contract describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realized by a combination of the resulting local systems, on the basis of appropriate infrastructure support.

The WS-CDL model [22] describes the participants of a composite Web Service, their role types and the relationships between the parties. It also contains a description of the information exchanged, the channels used for communication and the visible information of the different roles.

The main elements of a WS-CDL description are choreographies, which are defined in a hierarchical way. There is a root choreography, and any choreography can perform other inner choreographies. Choreographies prescribe the common rules that govern the ordering of exchanged messages and the collaborative behavior of the different parties. They consist of three parts:

- **Choreography life-line**: This describes the progression of a collaboration. Initially, the collaboration is established between the parties; then, some work is performed within it, and finally it completes either normally or abnormally.
- **Choreography exception block**: This specifies the additional interactions that should occur when a Choreography behaves in an abnormal way.
- **Choreography finalizer block**: This describes how to specify additional interactions that should occur to modify the effect of an earlier successfully completed Choreography (for example to confirm or undo the effect).

Each of these parts basically contains one activity, which describes the work to be done. There are basic activities (which perform the lowest level actions) and ordering structures. Basic activities are used to assign the variable values, or to indicate that a role type is performing some internal (non-visible) actions. Other basic activities supported by WS-CDL are the *interaction activities*, which describe the exchange of information between parties and the possible synchronizations of their observable information changes and the actual values of the exchanged information. They can be assigned a *time-out*, i.e., a time to be completed. When this time expires (after the interaction was initiated), if the interaction has not completed, the timeout occurs and the interaction finishes abnormally, causing an exception block to be executed in the choreography.

The ordering structures are the sequential composition, the choice and the parallel composition, so they combine activities with other ordering structures in a nested structure. There is also another class of activity supported by WS-CDL, the so-called workunits, which allow the execution of some activities when a certain condition holds. Workunits also permit the iteration of activities.

Time information in WS-CDL can appear both in the interactions (*time-outs*) and also in date/time variables (using XPath). These variables can be used in particular to delay the execution for a certain time, or they can be used to establish the instants at which some actions must be executed. For that purpose we may use the guards of workunits, by including in a guard an expression related to the value of a time variable.

4. Operational semantics

We first introduce the algebraic language that serves us as a metamodel of WS-CDL, and that allows us to define an operational semantics for it. In this model, for simplicity, we only consider WS-CDL documents with a single choreography (the *root* choreography), and we use indistinctly the terms *roles* and *parties* for describing the participants of a composite Web Service. For the choreography life-line we use the main WS-CDL activity constructions, both basic and structured. The same activity constructions are used in the exception block of the choreography. Observe that as a consequence of assuming a single choreography, we can have neither *finalizer blocks*, nor *perform* activities.

We are, therefore, considering a very relevant subset of WS-CDL, which includes its main structural constructions, integer variables, exception blocks and timing restrictions.

Table 1
Conversion table.

WS-CDL Syntax	Metamodel
<assign roleType="r"> <copy name="CName"> <source expression="n"/> <target variable="cdl:getVariable('v','','')" /> </copy> </assign>	assign(r,v,n)
<noAction roleType="r"/> or <silentAction roleType="r"/>	noaction(r)
<interaction name="name" ... <participate relationshipType="rname" fromRoleTypeRef="r1" toRoleTypeRef="r2" /> <exchange name="Cname" ... action="request"> <send variable="cdl:getVariable('v1','','')" ... > <receive variable="cdl:getVariable('v2','','')" ... > </exchange> <timeout time-to-complete="t" ... > ... </interaction>	inter(r1,r2,v1,v2,t)
<sequence> $activity_1$ $activity_2$ </sequence> | <choice> $activity_1$ $activity_2$ </choice> | <parallel> $activity_1$ $activity_2$ </parallel>	$\frac{activity_1 \; ; \; activity_2}{activity_1 \; \square \; activity_2}$ $activity_1 \parallel activity_2$
<workunit name="Name" guard="g" repeat="g' " block="true | false"> Activity </workunit>	workunit(g,block,g',Activity) block = true | false
Choreography exception handling	fail
<choreography name="Name" ... Activity1 <exceptionBlock name="EName"> Activity2 </exceptionBlock> ... </choreography>	(Activity1, Activity2)

We call *Var* the set of variable names used in the choreography, the *clock* variable being one of these variables, which contains the current time, thus automatically increasing its value as time elapses. Furthermore, it can be used in the guard of workunits to delay the execution, as we mentioned in Section 3. We assume that each role type uses its own variable names, i.e., a variable name can only be used by a single role type,[1] excepting the *clock* variable, whose value can be considered as obtained from a time server. For simplicity we only consider integer variables, although it would not be problematic to extend this assumption to any number of data types. Furthermore, we also consider that each interaction only contains one exchange element, which is used to communicate the value of a variable from one role type to the other.

The specific algebraic language, then, that we use for the activities is defined by the following BNF-notation:

$$A ::= fail \mid assign(r,v,n) \mid noaction(r) \mid inter(r_1,r_2,v_1,v_2,t) \mid \; A;A \; \mid A \square A \mid A \parallel A \mid workunit(g,block,g',A)$$

where r, r_1, r_2 range over the roletypes of the choreography, $t \in \mathbb{N} \cup \{\infty\}$, v, v_1, v_2 range over *Var*, $n \in \mathbb{Z}$, g, g' are predicates that use the variable names in *Var*, and *block* is a boolean. Given a predicate g, we will call $Vars(g)$ the set of variables used in g, which may have been initialized or not when they are used.

The correspondence between the syntax of WS-CDL and our metamodel is shown in Table 1. The basic activities are *fail*, *assign*, *noaction* and *inter* operations; *fail* is used to raise an exception, the control flow is transferred to the exception block, and after that the choreography terminates. The *assign* operation is used to assign the variable v at role r to n, and it is

[1] Actually, WS-CDL does not allow the use of shared variables.

immediate, i.e., it does not take any time to complete; the *noaction* captures either a silent or internal operation at role r, and it is immediate too. The *inter* operation is used to capture an interaction between roles r_1 and r_2, with a time-out t (which can be infinite), where the value of variable v_2 in r_2 is assigned to the value of variable v_1 of r_1. If the time-out expires and the interaction has not been executed, then, the exception block of the choreography is executed, after which the choreography terminates. An interaction also fails when the variable v_1 in r_1 is unassigned.

The ordering structures are the sequence, choice, parallel and workunit operations. The workunit operator has the following interpretation: first, if some of the variables used in g are not available, or if g evaluates to false, then, depending on the *block* attribute the workunit is skipped or blocked until g is evaluated to true. When the guard evaluates to true, the activity inside the workunit is executed, and when it terminates, the repetition condition g' is evaluated. If some variable used in g' is not available or if g' is false, then, the workunit terminates, otherwise the activity inside it is executed again. The sequence and parallel operators have the usual interpretation.

Concerning the choice operator semantics, we can read its textual semantics in [22], which states that "only one activity of those involved in the choice can be executed, but when the choice has workunits with guard conditions, the first workunit whose guard condition is true must be executed". Thus, there is a prioritization by means of lexical ordering in this case. However, in the case of a choice having both guarded workunits and other activities as alternatives, it states that "the selection criteria for those activities are non-observable". This textual description is rather ambiguous, since in some cases lexical ordering is used, whereas in others any activity can be selected. We consider that lexical ordering is not the best way to prioritize the activities. This could be actually done by introducing specific priorities in the activities, which is the subject of research that we are currently undertaking (see [20]). In this paper, then, we consider the following approach, which is in our opinion the most natural and best matches the goals of a choreography: any activity of those enabled[2] in the choice can be executed. We also impose the condition for the block attribute of the workunits which are alternatives of a choice that it be true, since in this case we need only consider those workunits whose guard evaluates to true, and abandoning the choice when a guard of a workunit is false would be pointless.

A choreography is now defined as a pair (A_1, A_2), where A_1 and A_2 are activities defined by the previous syntax. A_1 is the activity of the *life-line* of the choreography and A_2 is the activity of its exception block, which can be empty (denoted by $\emptyset$), because the exception block is optional. We do not consider a separate finalizer activity, because it can be part of A_1 (concatenated with it by a sequence operator).

We now introduce the operational semantics for this language, by using both *overbarred* and *underbarred* dynamic terms, which are used to capture the current state of the choreography throughout its execution.

Before introducing the dynamic terms, we need to consider an extended version of the activity syntax, in which we add the following operator $dinter(r_1, r_2, v_1, v_2, t, t')$, with $t' \leq t$, called *dynamic interaction*, which represents an interaction that initially had a time-out t and now has t' time units left before expiration. We will use letters $B, B_1, B_2, \ldots$ to denote activities with the extended syntax, which are used to define the *dynamic terms*, these are defined by the following BNF-notation:

$$D ::= \quad \overline{B} \quad | \quad \underline{B} \quad | \quad D ; B \quad | \quad B ; D \quad | \quad D \,\square\, B \quad | \quad B \,\square\, D \quad | D \parallel D \quad | \quad workunit(g, block, g', D)$$

The set of dynamic terms will be called *Dterms*.

The *overbars* are used to indicate that the corresponding term can initiate its execution, whereas *underbarred* terms have already finished their execution. Thus, as the activity evolves along its execution the bars are moving throughout the term syntax.

Example 1. Consider the activity $A = workunit(g, true, g', assign(r, v, 1))$. Its execution starts with the dynamic term $\overline{A}$, from which the guard g is evaluated. If all the variables in g are available, and g becomes true, then, we reach the dynamic term $D_1 = workunit(g, true, g', \overline{assign(r, v, 1)})$, which means that the assignment of v can now start at role r. Otherwise, if some variable needed to evaluate g is not available, or if g is false, as the *block* condition is *true*, the activity blocks until g changes its value to true. Once the assignment of v is done, the following dynamic term is reached: $D_2 = workunit(g, true, g', \underline{assign}(r, v, 1))$, from which g' is evaluated. If some variables needed to evaluate g' are not available or g' is false, then, the workunit ends and the dynamic term $\underline{A}$ is reached. Otherwise, when g' is true, D_1 is reached again.

In this example we have used dynamic terms to represent the current state of the system. However, dynamic terms like $\overline{B_1 \,\square\, B_2}$, $\overline{B_1} \,\square\, B_2$ and $B_1 \,\square\, \overline{B_2}$ correspond to the same state in the system, a state in which any alternative of the choice must be enabled. This means that in some cases the bars can be redistributed on a dynamic term yielding to an equivalent state. Thus, we now define the equivalence relation $\equiv$, as the **least equivalence relation** satisfying the rules of Table 2. By means of this equivalence relation we can identify those dynamic terms that can be obtained by moving backwards or forwards the bars on the terms without executing any action and which correspond to the same state in the system. It will also identify the activation of an interaction with the corresponding dynamic interaction that has the whole time-out to complete.

For any dynamic term D we will denote the class of dynamic terms equivalent to D by $[D]_{\equiv}$, and the set of classes of dynamic terms will be called *CDterms*.

[2] In the sense that it can execute some action at the current instant.

Table 2
Equivalence rules.

(Seq1)	$\overline{B_1; B_2} \equiv \overline{B_1}; B_2$	**(Seq2)**	$\underline{B_1}; B_2 \equiv B_1; \overline{B_2}$
(Seq3)	$B_1; \underline{B_2} \equiv \underline{B_1; B_2}$	**(Cho1)**	$\overline{B_1 \,\square\, B_2} \equiv \overline{B_1} \,\square\, B_2$
(Cho2)	$\overline{B_1 \,\square\, B_2} \equiv B_1 \,\square\, \overline{B_2}$	**(Cho3)**	$\underline{B_1} \,\square\, B_2 \equiv \underline{B_1 \,\square\, B_2}$
(Cho4)	$B_1 \,\square\, \underline{B_2} \equiv \underline{B_1 \,\square\, B_2}$	**(Par1)**	$\overline{B_1 \parallel B_2} \equiv \overline{B_1} \parallel \overline{B_2}$
(Par2)	$\underline{B_1} \parallel \underline{B_2} \equiv \underline{B_1 \parallel B_2}$		
(Inter)	$\overline{inter(r_1, r_2, v_1, v_2, t)} \equiv \overline{dinter(r_1, r_2, v_1, v_2, t, t)}$		
(Cong1)	$\dfrac{\forall op \in \{;, \square\},\quad D_1 \equiv D_2}{B\, op\, D_1 \equiv B\, op\, D_2\,,\quad D_1\, op\, B \equiv D_2\, op\, B}$		
(Cong2)	$\dfrac{D_1 \equiv D_2}{D \parallel D_1 \equiv D \parallel D_2\,,\quad D_1 \parallel D \equiv D_2 \parallel D}$		
(Cong3)	$\dfrac{D_1 \equiv D_2}{workunit(g, block, g', D_1) \equiv workunit(g, block, g', D_2)}$		

The rules of Table 2 are very intuitive in general. *Seq1* is used to activate the first activity of a sequence when the sequence becomes activated, *Seq2* allows us to activate B_2 when B_1 terminates, and *Seq3* establishes that once B_2 ends, the sequence $B_1; B_2$ ends too. *Cho1* and *Cho2* allow us to activate either alternative of a choice, while *Cho3* and *Cho4* establish that once the selected alternative terminates the choice itself ends too. *Par1* is used to activate both arguments in a parallel activity, and *Par2* establishes that, when both argument activities terminate, the parallel activity terminates, too. *Inter* identifies the activation of an interaction with the dynamic interaction having its whole time-out to be executed.

Lemma 1. *The relation $\equiv$ defined as the least equivalence relation fulfilling the rules in Table 2 is a congruence.*

Proof. $\equiv$ is defined as the least reflexive, symmetric, and transitive relation fulfilling the rules in Table 2. It is immediate that such a relation exists, and also that it is a congruence, due to rules *Cong1, Cong2* and *Cong3*. □

The following definition introduces the so-called *initial* and *final* dynamic terms, which are those dynamic terms that are equivalent to an overbarred (underbarred) extended activity.

Definition 1 (*Initial and final dynamic terms*). Given a dynamic term D, we say that D is initial (resp. final), denoted by $init(D)$ (resp. $final(D)$), when there exists an extended activity B such that $\overline{B} \in [D]_{\equiv}$ (resp. $\underline{B} \in [D]_{\equiv}$). In such a case we will say that the class $[D]_{\equiv}$ is initial (resp. final) too.

According to this definition, $(\overline{assign(r, v, n)})$, $(\overline{assign(r, v, n)} \,\square\, noaction(r))$ and $(\overline{assign(r, v, n)} \parallel \overline{noaction(r)})$ are initial, but not $(\underline{assign(r, v, n)})$, $((assign(r, v, n); \overline{assign(r', v', n')}) \,\square\, noaction(r))$ or $(\underline{assign(r, v, n)} \parallel \overline{fail})$. Similar examples can be written for final dynamic terms.

A choreography is executed within the context of the variables defined in it. We now define the *context* of a choreography, which captures which variables are available at the current instant, and their current values.

Definition 2 (*Context*). Given a choreography $C = (A_1, A_2)$, with roletypes $\mathcal{R}$ and variables *Var*, we define a *context* of C as a function $\mu : Var \to \mathbb{Z} \cup \{\epsilon\}$. Unavailable variables are assigned the ϵ value, otherwise this function provides us with the current value of the variable.

We denote the set of possible contexts of a choreography by *Contexts*. The *initial context*, denoted by μ_0, is that defined by assigning ϵ to all the variables in the choreography, except the *clock*, which is assigned to 0:

$$\mu_0(v) = \epsilon \quad \forall v \in Var \setminus \{clock\} \qquad \mu_0(clock) = 0$$

Given a context μ, a variable v and an integer value n, we denote by $\mu[v/n]$ the context obtained from μ by changing the value of v to n:

$$\mu[v/n](v') = \begin{cases} \mu(v') & \text{if } v' \neq v \\ n & \text{if } v' = v \end{cases}$$

We will also use this definition for n being an integer arithmetic expression that uses some variables of the choreography, with the natural interpretation, the value of v is replaced by the resulting value of n.

Now, given a predicate g and a context μ, we will write $sat(\mu, g)$ when $\forall v \in Vars(g)$, $\mu(v) \neq \epsilon$, and g evaluates to true under μ.

In order to capture the passage of time we need the following function, which ages a class of dynamic terms in one time unit:

Definition 3 (*Aging function*). The function $aging : CDterms \to CDterms$ is defined in a structural way, as follows:
For any dynamic terms D, D_1, D_2:

(1) If $final(D)$, then $aging([D]_{\equiv}) = [D]_{\equiv}$.
(2) $aging([\overline{fail}]_{\equiv}) = [\overline{fail}]_{\equiv}$.
(3) $aging([\overline{assign(r,v,n)}]_{\equiv}) = [\overline{assign(r,v,n)}]_{\equiv}$.
(4) $aging([\overline{noaction(r)}]_{\equiv}) = [\overline{noaction(r)}]_{\equiv}$.
(5) For $t' > 0$:$aging([\overline{dinter(r_1,r_2,v_1,v_2,t,t')}]_{\equiv}) = [\overline{dinter(r_1,r_2,v_1,v_2,t,t'-1)}]_{\equiv}$, where we take $\infty - 1 = \infty$.
(6) $aging([\overline{dinter(r_1,r_2,v_1,v_2,t,0)}]_{\equiv}) = [\overline{fail}]_{\equiv}$.
(7) $aging([\overline{workunit(g,block,g',B)}]_{\equiv}) = [\overline{workunit(g,block,g',B')}]_{\equiv}$, with B' such that $\overline{B'} \in aging([\overline{B}]_{\equiv})$.
(8) $aging([workunit(g,block,g',D)]_{\equiv}) = [workunit(g,block,g',D')]_{\equiv}$, with $D' \in aging([D]_{\equiv})$.
(9) If $\neg final(D)$: $aging([D;B]_{\equiv}) = [D';B]_{\equiv}$, and $aging([B;D]_{\equiv}) = [B;D']_{\equiv}$, with $D' \in aging([D]_{\equiv})$.
(10) If $\neg init(D) \wedge \neg final(D)$: $aging([B \Box D]_{\equiv}) = [B \Box D']_{\equiv}$, and $aging([D \Box B]_{\equiv}) = [D' \Box B]_{\equiv}$, with $D' \in aging([D]_{\equiv})$.
(11) If $init(D) \wedge \neg final(D)$: $aging([B \Box D]_{\equiv}) = [B' \Box D']_{\equiv}$, and $aging([D \Box B]_{\equiv}) = [D' \Box B']_{\equiv}$, with $D' \in aging([D]_{\equiv})$ and B' such that $\overline{B'} \in aging([\overline{B}]_{\equiv})$.
(12) If $\neg final(D_1) \wedge \neg final(D_2)$: $aging([D_1 \parallel D_2]_{\equiv}) = [D'_1 \parallel D'_2]_{\equiv}$, with $D'_1 \in aging([D_1]_{\equiv})$ and $D'_2 \in aging([D_2]_{\equiv})$.
(13) If $final(D_1) \wedge \neg final(D_2)$: $aging([D_1 \parallel D_2]_{\equiv}) = [D_1 \parallel D'_2]_{\equiv}$, and $aging([D_2 \parallel D_1]_{\equiv}) = [D'_2 \parallel D_1]_{\equiv}$, with $D'_2 \in aging([D_2]_{\equiv})$.

From this definition we can see that when an interaction expires (point 6) we obtain a failure, which will allow us to execute the exception block (except if we find ourselves in a choice with some other possible alternatives, as we will see later). The passage of time for dynamic interactions is captured by means of point 5. We can also see that the passage of time over an activated workunit is passed to the activity inside it (point 7), since we consider that the first activity of the workunit is in some sense *activated* once the workunit has been reached (although it can only be executed when the guard condition is true). Point 11 also requires some explanations, in this case the passage of time over an activated choice is passed to both argument activities. As the remaining points are quite self-explanatory, we shall omit further comment.

Proposition 1. *The function aging is well defined.*

Proof. The first point establishes that no time elapses for final dynamic terms. All the remaining cases cover the different possibilities we may have for non-final dynamic terms. Points 2–6 are obvious. Now observe that when the function *aging* is applied to an initial class, the resulting class is initial too (all points of the definition hold this, so a simple structural induction can be used to prove this fact). From this fact point 7 becomes clear, and we can guarantee that there exists B' such that $\overline{B'} \in aging\left([\overline{B}]_{\equiv}\right)$, and, in fact, it is unique (we simply remove the bar from the term). Point 8 is also well defined, this is a consequence of rule *Cong3* of Table 2, we can take either representative D' of the class, and the result will always be the same. The same occurs for the last points of the definition, where rules *Cong1* and *Cong2* are used to guarantee that the resulting class is always the same, whichever the chosen representative.

Notice also that this definition covers all the possible cases for the syntax of *CDterms*, taking into account the $\equiv$-equivalence. □

With the function *aging* we transform one class into another, capturing the elapse of one time unit. However, in some cases we do not allow the passage of time, since some movement must be made immediately. This occurs, for instance, when an exception has been raised; in this case the exception block is immediately executed. Furthermore, in general, not only time elapsing, but all the possible evolutions of a class depend on the current context. Hence, we introduce the following definition.

Definition 4 (*Contextual activity terms*). A *contextual activity term* is a pair $([D]_{\equiv}, \mu)$, where D is a dynamic term and μ a context.

We now define a boolean function *elapse*, which indicates whether time can elapse or not for any contextual activity term.

Definition 5 (*Function elapse*). The function $elapse : CDterms \times Contexts \to Boolean$ is defined in a structural way, as follows:
For any dynamic terms D, D_1, D_2 and any context μ:

(1) If $final(D)$, then $elapse([D]_{\equiv}, \mu) = true$.
(2) $elapse([\overline{fail}]_{\equiv}, \mu) = false$.

(3) $elapse(\overline{[assign(r,v,n)]}_{\equiv},\mu) = true$.

(4) $elapse(\overline{[noaction(r)]}_{\equiv},\mu) = true$.

(5) If $\mu(v_1) \neq \epsilon$: $elapse(\overline{[dinter(r_1,r_2,v_1,v_2,t,t')]}_{\equiv},\mu) = true$.

(6) If $\mu(v_1) = \epsilon$: $elapse(\overline{[dinter(r_1,r_2,v_1,v_2,t,t')]}_{\equiv},\mu) = false$.

(7) $elapse(\overline{[workunit(g,block,g',B)]}_{\equiv},\mu) = block$.

(8) If $\neg final(D)$: $elapse([workunit(g,block,g',D)]_{\equiv},\mu) = elapse([D]_{\equiv})$.

(9) If $final(D)$: $elapse([workunit(g,block,g',D)]_{\equiv},\mu) = false$.

(10) If $\neg final(D)$: $elapse([D;B]_{\equiv},\mu) = elapse([B;D]_{\equiv},\mu) = elapse([D]_{\equiv},\mu)$.

(11) If $\neg init(D) \wedge \neg final(D)$: $elapse([B \,\square\, D]_{\equiv},\mu) = elapse([D \,\square\, B]_{\equiv},\mu) = elapse([D]_{\equiv},\mu)$.

(12) If $init(D)$: $elapse([B \,\square\, D]_{\equiv},\mu) = elapse([D \,\square\, B]_{\equiv},\mu) = elapse([\overline{B}]_{\equiv},\mu) \vee elapse([D]_{\equiv},\mu)$.

(13) If $\neg final(D_1) \wedge \neg final(D_2)$:$elapse([D_1 \parallel D_2]_{\equiv},\mu) = elapse([D_1]_{\equiv},\mu) \wedge elapse([D_2]_{\equiv},\mu)$.

(14) If $final(D_1) \wedge \neg final(D_2)$: $elapse([D_1 \parallel D_2]_{\equiv},\mu) = elapse([D_2 \parallel D_1]_{\equiv},\mu) = elapse([D_2]_{\equiv},\mu)$.

Table 3
Transition rules for contextual activity terms (I).

(Clock)	$\dfrac{elapse([D]_{\equiv},\mu)}{([D]_{\equiv},\mu) \longrightarrow_1 (aging([D]_{\equiv}),\mu[clock/clock+1])}$
(Fail)	$\dfrac{}{(\overline{[fail]}_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv},\mu)}$
(Assign)	$\dfrac{}{(\overline{[assign(r,v,n)]}_{\equiv},\mu) \xrightarrow{assign(r,v,n)} (\underline{[assign(r,v,n)]}_{\equiv},\mu[v/n])}$
(Noact)	$\dfrac{}{(\overline{[noaction(r)]}_{\equiv},\mu) \xrightarrow{noaction(r)} (\underline{[noaction(r)]}_{\equiv},\mu)}$
(Int1)	$\dfrac{\mu(v_1) \neq \epsilon}{(\overline{[dinter(r_1,r_2,v_1,v_2,t,t')]}_{\equiv},\mu) \xrightarrow{inter(r_1,r_2,v_1,v_2,t)} (\underline{[dinter(r_1,r_2,v_1,v_2,t,t')]}_{\equiv},\mu[v_2/v_1])}$
(Int2)	$\dfrac{\mu(v_1) = \epsilon}{(\overline{[dinter(r_1,r_2,v_1,v_2,t,t')]}_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv},\mu)}$
(Work1)	$\dfrac{sat(\mu,g),\ (\overline{[B]}_{\equiv},\mu) \xrightarrow{a} ([D]_{\equiv},\mu'),\ a \neq fail}{(\overline{[workunit(g,block,g',B)]}_{\equiv},\mu) \xrightarrow{a} (workunit(g,block,g',[D]_{\equiv}),\mu')}$
(Work2)	$\dfrac{sat(\mu,g),\ (\overline{[B]}_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv},\mu)}{(\overline{[workunit(g,block,g',B)]}_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv}),\mu)}$
(Work3)	$\dfrac{\neg sat(\mu,g)}{(\overline{[workunit(g,false,g',B)]}_{\equiv},\mu) \xrightarrow{\emptyset} (\underline{[workunit(g,false,g',B)]}_{\equiv},\mu)}$
(Work4)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ a \neq fail}{([workunit(g,block,g',D)]_{\equiv},\mu) \xrightarrow{a} ([workunit(g,block,g',D')]_{\equiv},\mu')}$
(Work5)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv},\mu)}{([workunit(g,block,g',D)]_{\equiv},\mu) \xrightarrow{fail} (\underline{[fail]}_{\equiv},\mu)}$
(Work6)	$\dfrac{sat(\mu,g'),\ D \equiv \underline{B}}{([workunit(g,block,g',D)]_{\equiv},\mu) \xrightarrow{\emptyset} ([workunit(g,block,g',\overline{B})]_{\equiv},\mu)}$
(Work7)	$\dfrac{\neg sat(\mu,g'),\ D \equiv \underline{B}}{([workunit(g,block,g',D)]_{\equiv},\mu) \xrightarrow{\emptyset} (\underline{[workunit(g,block,g',B)]}_{\equiv},\mu)}$

Table 4
Transition rules for contextual activity terms (II).

(Seq1–2)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ a \neq fail}{([D;B]_{\equiv},\mu) \xrightarrow{a} ([D';B]_{\equiv},\mu')}$ $\qquad$ $\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ a \neq fail}{([B;D]_{\equiv},\mu) \xrightarrow{a} ([B;D']_{\equiv},\mu')}$
(Seq3–4)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),}{([D;B]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$ $\qquad$ $\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),}{([B;D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$
(Choi1–2)	$\dfrac{([\overline{B_1}]_{\equiv},\mu) \xrightarrow{a} ([D]_{\equiv},\mu'),\ a \neq fail}{([\overline{B_1 \,\square\, B_2}]_{\equiv},\mu) \xrightarrow{a} ([D \,\square B_2]_{\equiv},\mu')}$ $\qquad$ $\dfrac{([\overline{B_2}]_{\equiv},\mu) \xrightarrow{a} ([D]_{\equiv},\mu'),\ a \neq fail}{([\overline{B_1 \,\square\, B_2}]_{\equiv},\mu) \xrightarrow{a} ([B_1 \,\square D]_{\equiv},\mu')}$
(Choi3)	$\dfrac{([\overline{B_1}]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),\ ([\overline{B_2}]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}{([\overline{B_1 \,\square\, B_2}]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$
(Choi4)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ \neg init(D),\ a \neq fail}{([D \,\square\, B]_{\equiv},\mu) \xrightarrow{a} ([D' \,\square B]_{\equiv},\mu')}$
(Choi5)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ \neg init(D),\ a \neq fail}{([B \,\square\, D]_{\equiv},\mu) \xrightarrow{a} ([B \,\square D']_{\equiv},\mu')}$
(Choi6–7)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),\ \neg init(D)}{([B \,\square\, D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$ $\qquad$ $\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),\ \neg init(D)}{([D \,\square\, B]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$
(Par1)	$\dfrac{([D_1]_{\equiv},\mu) \xrightarrow{a} ([D'_1]_{\equiv},\mu'),\ a \neq fail,\ ([D_2]_{\equiv},\mu) \xrightarrow{fail}\!\!\!\!\!\!/\ \ }{([D_1 \| D_2]_{\equiv},\mu) \xrightarrow{a} ([D'_1 \| D_2]_{\equiv},\mu')}$
(Par2)	$\dfrac{([D_2]_{\equiv},\mu) \xrightarrow{a} ([D'_2]_{\equiv},\mu'),\ a \neq fail,\ ([D_1]_{\equiv},\mu) \xrightarrow{fail}\!\!\!\!\!\!/\ \ }{([D_1 \| D_2]_{\equiv},\mu) \xrightarrow{a} ([D_1 \| D'_2]_{\equiv},\mu')}$
(Par3–4)	$\dfrac{([D_2]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}{([D_1 \| D_2]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$ $\qquad$ $\dfrac{([D_1]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}{([D_1 \| D_2]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}$

It is immediate to check that *elapse* is a well defined function. By means of elapse the passage of time is not allowed when an exception has been raised (point 2), except in the case of the failure being caused by an alternative of a choice, since some other alternatives could be allowed. Thus, for instance, if an interaction with a time-out has expired, this interaction cannot be executed, but there may be some other possible alternatives in the choice that are still enabled. In point 6 we can also see that when the source variable of an interaction is unassigned time cannot elapse, because we immediately raise an exception. In the case of an activated workunit (point 7), depending on the *block* attribute we can wait or not, and when the activity of the workunit terminates, the repetition condition g' must be evaluated immediately, so no time can elapse here (point 9). For an activated choice (point 12) we allow the passage of time when at least one alternative does allow it. Thus, in a choice we may have some interactions with time-outs that have expired, but the choice may still offer some alternatives. However, for the parallel operator, time can only elapse when both alternatives allow the passage of time.

Definition 6. We define a *dynamic choreography term* as a pair of one of the following forms: $([D]_{\equiv}, A_2)$ or $(A_1, [D]_{\equiv})$, where $[D]_{\equiv}$ corresponds to the activity in execution in the choreography (the life-line or its exception block), and A_2 can be empty.

We also define a *contextual dynamic choreography term*, as a pair $(\mathcal{C}, \mu)$, where $\mathcal{C}$ is a dynamic choreography term and μ is a context.

Given a choreography $C = (A_1, A_2)$, the *initial contextual dynamic term* of C is[3] $([\overline{A_1}]_{\equiv}, A_2, \mu_0)$.

In Tables 3 and 4, we introduce the rules that define the transitions for the contextual activity terms, where we can see that we have two types of transition:

[3] We will write contextual dynamic choreography terms as triples, by omitting the parentheses for the dynamic choreography term.

Table 5
Transition rules for choreographies.

(Cor1)	$\dfrac{([D]_{\equiv},\mu) \longrightarrow_1 ([D']_{\equiv},\mu')}{([D]_{\equiv},A_2,\mu) \longrightarrow_1 ([D']_{\equiv},A_2,\mu')}$	**(Cor2)**	$\dfrac{([D]_{\equiv},\mu) \longrightarrow_1 ([D']_{\equiv},\mu')}{(A_1,[D]_{\equiv},\mu) \longrightarrow_1 (A_1,[D']_{\equiv},\mu')}$
(Cor3)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ a \neq fail}{([D]_{\equiv},A_2,\mu) \xrightarrow{a} ([D']_{\equiv},A_2,\mu')}$	**(Cor4)**	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu'),\ a \neq fail}{(A_1,[D]_{\equiv},\mu) \xrightarrow{a} (A_1,[D']_{\equiv},\mu')}$
(Cor5)	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu),\ A_2 \neq \emptyset}{([D]_{\equiv},A_2,\mu) \xrightarrow{fail} (B_1,[\overline{A_2}]_{\equiv},\mu)}$	**(Cor6)**	$\dfrac{([D]_{\equiv},\mu) \xrightarrow{fail} ([\underline{fail}]_{\equiv},\mu)}{(A_1,[D]_{\equiv},\mu) \xrightarrow{fail} (A_1,[\underline{fail}]_{\equiv},\mu)}$

- $([D]_{\equiv},\mu) \longrightarrow_1 ([D']_{\equiv},\mu')$: which represents the passage of one time unit.
- $([D]_{\equiv},\mu) \xrightarrow{a} ([D']_{\equiv},\mu')$: which represents the execution of some basic activity a or an empty movement (denoted by $a = \emptyset$). In this case no time elapses.

In rules *Par1* and *Par2* of Table 4 we use the notation $([D]_{\equiv},\mu) \xrightarrow{fail}\!\!\!\!\!/\ \ $ to mean that no transition labelled with *fail* can be executed from $([D]_{\equiv},\mu)$.

Let us see the informal interpretation of these rules:

- Rule *Clock* is used to capture the passage of one time unit, where both functions, *elapse* and *aging*, are used for that purpose.
- Rules *Fail*, *Assign* and *Noact* are evident. *Int1* captures the execution of an activated interaction, when the source variable has a value assigned. Otherwise, rule *Int2* is used to raise an exception.
- Rules *Work1* to *Work7* establish the semantics of workunits. For a workunit whose guard condition evaluates to true, we can execute any initial movement of the activity inside it (rule *Work1*). Once the workunit is activated, if the activity inside the workunit can execute a *fail* movement, we immediately raise an exception (rules *Work2* and *Work5*). When the *block* attribute is false, and the guard condition is not fulfilled,[4] the workunit is skipped at once (rule *Work3*). The execution of the activities inside the workunit is captured by rule *Work4*. Rule *Work6* allows us to restart the activity inside the workunit when it has finished and the repetition condition holds, whereas rule *Work7* is used to abandon the workunit when that condition does not hold.
- Rules *Seq1* to *Seq4* capture the semantics of the sequence operator, while *Choi1* to *Choi7* define the semantics of the choice. The rules for the sequence are highly intuitive, so we omit an explanation about them. In the case of the choice operator, *Choi1* and *Choi2* are used to resolve the choice when one argument activity can execute a movement (different from *fail*). Once the choice has been decided by executing a movement of one of its argument activities, this activity continues executing until completion (*Choi4–5*). If the activity in execution can make a *fail* movement, an exception is raised (rules *Choi6–7*). In rule *Choi3* we can see that the choice can only execute a *fail* movement when both arguments are able to do that. Accordingly, when an alternative fails (for instance, a time-out of an interaction has expired), this alternative is not considered for execution, but the other ones can proceed (in fact, we allow time elapsing in that case, because we may have some other interactions that can be executed some time later).
- Finally, rules *Par1–2* capture the (independent) parallel execution of the argument activities of a parallel operator, and *Par3–4* are used to raise an exception when one component is able to do so.

The rules for choreographies are those introduced in Table 5, which capture the evolution of contextual dynamic choreography terms. *Cor1–2* allow the passage of time on the choreography activity in execution, whereas *Cor3–4* allow the evolution of the activity in execution, except in the case of failure. In that case, rules *Cor5–6* are used, the first to activate the activity of the exception block, and the second to terminate the activity of the exception block when it fails. In rule *Cor5* the term B_1 is that obtained by removing the bars on D.

Definition 7 (*Labelled transition system*). For any contextual activity term $([D]_{\equiv},\mu)$ we define its labelled transition system, denoted by $lts([D]_{\equiv},\mu)$, as that obtained by the application of the rules in Tables 3 and 4, starting from $([D]_{\equiv},\mu)$: $lts([D]_{\equiv},\mu) = (Q, q_0, \rightarrow)$, where Q is the set of contextual activity terms that are reachable by using the rules in Tables 3 and 4, starting from $q_0 = ([D]_{\equiv},\mu)$, and $\rightarrow = \rightarrow_1 \cup \{\xrightarrow{a} \mid \text{for all basic activity } a, \text{ or } a = \emptyset\}$.

Then, for any choreography $C = (A_1, A_2)$, we define the semantics of C as the labelled transition transition system obtained by the application of rules in Table 5 for the initial contextual dynamic choreography term of C, $c_0 = ([\overline{A_1}]_{\equiv}, A_2, \mu_0)$:

$$lts(C) = (\mathcal{Q}, c_0, \rightarrow)$$

where $\mathcal{Q}$ is the set of contextual dynamic choreography terms that are reachable by the rules in Table 5, starting from c_0, and $\rightarrow = \rightarrow_1 \cup \{\xrightarrow{a} \mid \text{for all basic activity } a, \text{ or } a = \emptyset\}$.

[4] This case cannot occur as alternative of a choice, due to the syntactical restriction introduced.

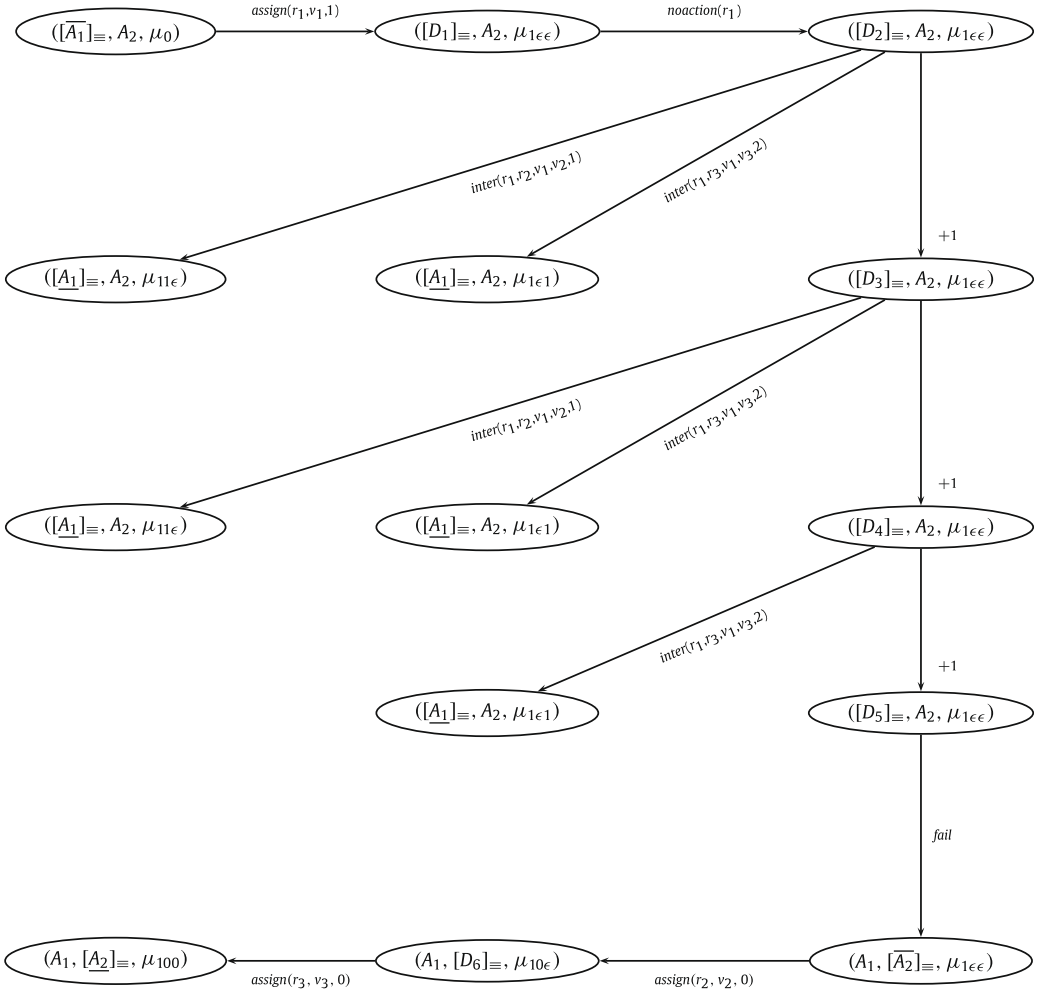

Fig. 1. A piece of $lts(C)$.

Example 2. Let us consider the choreography $C = (A_1, A_2)$, where

$$A_1 = assign(r_1, v_1, 1); noaction(r_1); (inter(r_1, r_2, v_1, v_2, 1) \square inter(r_1, r_3, v_1, v_3, 2))$$
$$A_2 = assign(r_2, v_2, 0); assign(r_3, v_3, 0)$$

Then, in Fig. 1 we show a piece of the labelled transition system of C, where

$$D_1 = assign(r_1, v_1, 1); \overline{noaction(r_1)}; (inter(r_1, r_2, v_1, v_2, 1) \square inter(r_1, r_3, v_1, v_3, 2))$$
$$D_2 = assign(r_1, v_1, 1); noaction(r_1); \overline{(dinter(r_1, r_2, v_1, v_2, 1, 1) \square dinter(r_1, r_3, v_1, v_3, 2, 2))}$$
$$D_3 = assign(r_1, v_1, 1); noaction(r_1); \overline{(dinter(r_1, r_2, v_1, v_2, 1, 0) \square dinter(r_1, r_3, v_1, v_3, 2, 1))}$$
$$D_4 = assign(r_1, v_1, 1); noaction(r_1); \overline{(fail \square dinter(r_1, r_3, v_1, v_3, 2, 0))}$$
$$D_5 = assign(r_1, v_1, 1); noaction(r_1); \overline{(fail \square fail)}$$
$$D_6 = assign(r_2, v_2, 0); \overline{assign(r_3, v_3, 0)}$$

and

$$\mu_{ijk}(v_1) = i, \quad \mu_{ijk}(v_2) = j, \quad \mu_{ijk}(v_3) = k, \quad \mu_{ijk}(v_n) = \epsilon \quad \forall v_n \neq clock, v_1, v_2, v_3$$

Of course, Fig. 1 only shows some of the possible timed traces of this example. For instance, we have taken a timed trace in which $assign(r_1, v_1, 1)$ is executed at time 0, but it could also be the case that before its execution some time has elapsed, and the same occurs for some other states of the labelled transition system of C.

Example 3. Let us now consider the choreography $C = (A_1 \| A_2 \| A_3, \emptyset)$, where

$A_1 = assign(r_1, v_1, 1); workunit(v_1 + v_3 = 4, true, v_1 + v_2 = 3, inter(r_1, r_2, v_1, v_2, \infty))$
$A_2 = assign(r_2, v_2, 2)$
$A_3 = assign(r_3, v_3, 3); assign(r_3, v_3, 4)$

This choreography may behave correctly or not, depending on the order in which the different actions are executed. Let us see three different timed traces of it:

- $assign(r_1, v_1, 1).\ assign(r_3, v_3, 3).\ inter(r_1, r_2, v_1, v_2, \infty).\ assign(r_2, v_2, 2).\ assign(r_3, v_3, 4)$. With this trace the choreography ends correctly its execution, reaching the contextual dynamic choreography term $(\underline{[A_1 \| A_2 \| A_3]}_{\equiv}, \emptyset, \mu_{124})$.
- $assign(r_2, v_2, 2).\ assign(r_1, v_1, 1).\ assign(r_3, v_3, 3).\ inter(r_1, r_2, v_1, v_2, \infty).\ assign(r_3, v_3, 4)$. Again, it behaves correctly, reaching the contextual dynamic choreography term $(\underline{[A_1 \| A_2 \| A_3]}_{\equiv}, \emptyset, \mu_{114})$.
- $assign(r_3, v_3, 3).\ assign(r_3, v_3, 4).\ assign(r_2, v_2, 2).\ assign(r_1, v_1, 1)$. In this case the system becomes deadlocked, waiting for the interaction to occur, but it will never be executed, due to the values of the variables v_1 and v_3.

4.1. Bisimulation relation

We first observe that the simple notion of isomorphism between labelled transition systems is not a congruence in our model, as the following example illustrates:

Example 4. Let us consider the following dynamic terms:

$D_1 = \overline{assign(r_1, v_1, 1)}$
$D_2 = \overline{assign(r_1, v_1, 1)}; workunit(g, true, g', assign(r_2, v_2, 2))$

and

$D'_1 = \underline{assign(r_1, v_1, 1)}$
$D'_2 = assign(r_1, v_1, 1); \overline{workunit(g, true, g', assign(r_2, v_2, 2))}$

where g is the guard $v_2 = 3$ and g' is the guard $v_3 = 3$. We may then construct $lts([D_1]_{\equiv}, \mu_0)$ and $lts([D_2]_{\equiv}, \mu_0)$, as shown in Fig. 2.

Both labelled transition systems are isomorphic, but if we consider:

$D_3 = D_1; assign(r_3, v_3, 3)$
$D_4 = D_2; assign(r_3, v_3, 3)$

we obtain that $lts([D_3]_{\equiv}, \mu_0)$ is not isomorphic to $lts([D_4]_{\equiv}, \mu_0)$ (Fig. 3), where in this figure:

$D'_3 = D_1; \overline{assign(r_3, v_3, 3)}$
$D''_3 = D_1; \underline{assign(r_3, v_3, 3)}$
$D'_4 = assign(r_1, v_1, 1); \overline{workunit(g, true, g', assign(r_2, v_2, 2))}; assign(r_3, v_3, 3)$

In this specific case we could solve the problem by requiring for any couple of related dynamic term classes to fulfill the following condition: when one of these terms is final (or initial), the other one must be final (or initial) too. Unfortunately, this solution does not work in general, as the following example illustrates.

Example 5. Let us now consider:

$D_1 = \overline{assign(r_1, v_1, 1)}; workunit(g_1, true, g', assign(r_3, v_3, 3))$
$D_2 = \overline{assign(r_1, v_1, 1)}; workunit(g_2, true, g', assign(r_3, v_3, 3))$

where g_1 is the guard $v_1 = 3$, g_2 is the guard $v_1 = 2$ and g' is the guard $v_2 = 2$.

According to the previous criterion, $lts([D_1]_{\equiv}, \mu_0)$ and $lts([D_2]_{\equiv}, \mu_0)$ would be equivalent, in the sense that their labelled transition systems are isomorphic, and they fulfill the additional restriction introduced, since for every pair of related terms it follows that when one is initial (final) the other is initial (final), too.

However, if we now consider:

$D_3 = D_1 \| \overline{assign(r_1, v_1, 3)}$
$D_4 = D_2 \| \overline{assign(r_1, v_1, 3)}$

It is straightforward to prove that their corresponding labelled transition systems are not isomorphic, because D_3 would be able to execute the action inside the workunit: $assign(r_3, v_3, 3)$, whereas D_4 will not be able to execute its action $assign(r_3, v_3, 3)$.

The problem here is that we need to take into account not only the structure of the labelled transition systems, but also the contexts from which the terms evolve. We then introduce the following definition of bisimulation.

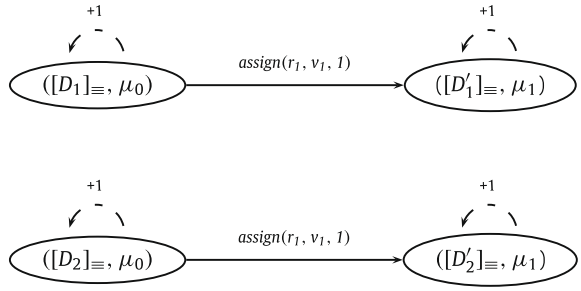

Fig. 2. $lts(([D_1]_{\equiv}, \mu_0))$ and $lts(([D_2]_{\equiv}, \mu_0))$.

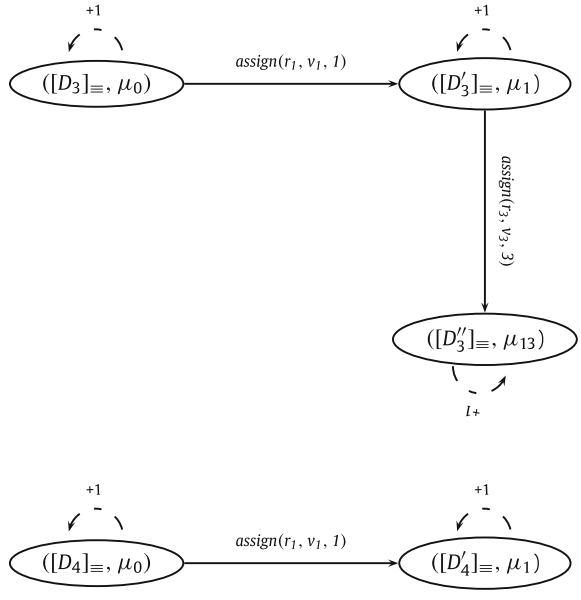

Fig. 3. $lts(([D_3]_{\equiv}, \mu_0))$ and $lts(([D_4]_{\equiv}, \mu_0))$.

Definition 8 (*Bisimulation relation*). A binary relation $\mathcal{R} \subseteq CDterms \times CDterms$ over the classes of dynamic terms is a bisimulation relation if $([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R}$ implies, for all contexts μ, μ', and for all basic activity a (or $a = \emptyset$):

(i) $final(D_1)$ if and only if $final(D_2)$.

(ii) $init(D_1)$ if and only if $init(D_2)$.

(iii) Whenever $([D_1]_{\equiv}, \mu) \xrightarrow{a} ([D'_1]_{\equiv}, \mu')$, then, for some D'_2, $([D_2]_{\equiv}, \mu) \xrightarrow{a} ([D'_2]_{\equiv}, \mu')$ and $([D'_1]_{\equiv}, [D'_2]_{\equiv}) \in \mathcal{R}$.

(iv) Whenever $([D_2]_{\equiv}, \mu) \xrightarrow{a} ([D'_2]_{\equiv}, \mu')$, then, for some D'_1, $([D_1]_{\equiv}, \mu) \xrightarrow{a} ([D'_1]_{\equiv}, \mu')$ and $([D'_1]_{\equiv}, [D'_2]_{\equiv}) \in \mathcal{R}$.

(v) Whenever $([D_1]_{\equiv}, \mu) \longrightarrow_1 ([D'_1]_{\equiv}, \mu')$, then, for some D'_2, $([D_2]_{\equiv}, \mu) \longrightarrow_1 ([D'_2]_{\equiv}, \mu')$ and $([D'_1]_{\equiv}, [D'_2]_{\equiv}) \in \mathcal{R}$.

(vi) Whenever $([D_2]_{\equiv}, \mu) \longrightarrow_1 ([D'_2]_{\equiv}, \mu')$, then, for some D'_1, $([D_1]_{\equiv}, \mu) \longrightarrow_1 ([D'_1]_{\equiv}, \mu')$ and $([D'_1]_{\equiv}, [D'_2]_{\equiv}) \in \mathcal{R}$.

Two classes of dynamic terms $[D_1]_{\equiv}$ and $[D_2]_{\equiv}$ are then bisimilar ($[D_1]_{\equiv} \sim [D_2]_{\equiv}$) if there exists a bisimulation relation $\mathcal{R}$ such that $([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R}$.

An immediate consequence of this definition is that for any $[D_1]_{\equiv}, [D_2]_{\equiv}$ such that $[D_1]_{\equiv} \sim [D_2]_{\equiv}$, it follows that $aging([D_1]_{\equiv}) \sim aging([D_2]_{\equiv})$.

Definition 9. We say that two activities (with the extended syntax) B_1 and B_2 are bisimilar, denoted by $B_1 \sim B_2$, if and only if $[\overline{B_1}]_{\equiv} \sim [\overline{B_2}]_{\equiv}$.

Theorem 1 (Congruence). *For any given activities* (*with the extended syntax*) B_1, B_2 *such that* $B_1 \sim B_2$, *and for any activity* B :

(**i**) $B_1; B \sim B_2; B$ *and* $B; B_1 \sim B; B_2$.

(**ii**) $B_1 \square B \sim B_2 \square B$ *and* $B \square B_1 \sim B \square B_2$.

(**iii**) $B_1 \| B \sim B_2 \| B$ *and* $B \| B_1 \sim B \| B_2$.
(**iv**) $workunit(g, block, g', B_1) \sim workunit(g, block, g', B_2)$.

Proof. Since $[\overline{B_1}]_{\equiv} \sim [\overline{B_2}]_{\equiv}$ there is a bisimulation relation $\mathcal{R}$ according to Definition 8. We can then define the corresponding bisimulation relation for each case, where in these definitions $Reach([\overline{B}]_{\equiv})$ will denote the classes of dynamic terms that we can obtain from $[\overline{B}]_{\equiv}$, considering any possible context all along its evolution. We define it recursively, as follows:

$$\begin{aligned} Reach([\overline{B}]_{\equiv}) = \; & \{[\overline{B}]_{\equiv}\} \cup \{[D]_{\equiv} \mid \exists D' \in Reach([\overline{B}]_{\equiv}), \text{ for some contexts } \mu, \mu' : \\ & ([D']_{\equiv}, \mu') \longrightarrow_1 ([D]_{\equiv}, \mu), \text{ or } ([D']_{\equiv}, \mu') \xrightarrow{a} ([D]_{\equiv}, \mu), \text{ for} \\ & \text{some action } a \text{ (or } a = \emptyset)\} \end{aligned}$$

We provide the bisimulation relation for each case, but we omit the proofs, since they are all straightforward applications of $\mathcal{R}$, Tables 3 and 4 and Definition 8.

(i) For $B_1; B \sim B_2; B$ we consider:

$$\begin{aligned} \mathcal{R}_{;} = \; & \{ ([D_1; B]_{\equiv}, [D_2; B]_{\equiv}) \; : \; ([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R}\} \cup \\ & \{([B_1; D]_{\equiv}, [B_2; D]_{\equiv}) \; : \; [D]_{\equiv} \in Reach([\overline{B}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\} \end{aligned}$$

The other case is defined analogously, and the same occurs in (ii) and (iii) for the symmetric results.

(ii) We now take:

$$\begin{aligned} \mathcal{R}_{\Box} = \; & \{ ([D_1 \Box B]_{\equiv}, [D_2 \Box B]_{\equiv}) \; : \; ([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R}\} \cup \\ & \{([B_1 \Box D]_{\equiv}, [B_2 \Box D]_{\equiv}) \; : \; [D]_{\equiv} \in Reach([\overline{B}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\} \end{aligned}$$

(iii) In this case:

$$\mathcal{R}_{\|} = \{ ([D_1 \| D]_{\equiv}, [D_2 \| D]_{\equiv}) : ([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R}, \; [D]_{\equiv} \in Reach([\overline{B}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$$

(iv) In this case we consider:

$$\begin{aligned} \mathcal{R}_{w} = \; & \{ ([\overline{workunit(g, block, g', B_1)}]_{\equiv}, [\overline{workunit(g, block, g', B_2)}]_{\equiv})\} \cup \\ & \{([workunit(g, block, g', D_1)]_{\equiv}, [workunit(g, block, g', D_2)]_{\equiv} \; : \; ([D_1]_{\equiv}, [D_2]_{\equiv}) \in \mathcal{R} \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\} \\ & \cup \{([\underline{workunit(g, block, g', B_1)}]_{\equiv}, [\underline{workunit(g, block, g', B_2)}]_{\equiv})\} \end{aligned}$$ □

We can also conclude the following algebraic properties:

Proposition 2. *For any activities (with the extended syntax)* B_1, B_2, B_3 :
(**i**) $B_1 \Box B_1 \sim B_1$.
(**ii**) $B_1 \Box B_2 \sim B_2 \Box B_1$.
(**iii**) $(B_1 \Box B_2) \Box B_3 \sim B_1 \Box (B_2 \Box B_3)$.
(**iv**) $B_1; (B_2 \Box B_3) \sim (B_1; B_2) \Box (B_1; B_3)$.
(**v**) $(B_1 \Box B_2); B_3 \sim (B_1; B_3) \Box (B_2; B_3)$.
(**vi**) $B_1 \| B_2 \sim B_2 \| B_1$.
(**vii**) $(B_1 \| B_2) \| B_3 \sim B_1 \| (B_2 \| B_3)$.

Proof. The bisimulation relations that we use for each case are:

(i) $\mathcal{R} = \{ ([D_1 \Box B]_{\equiv}, [D_1]_{\equiv}) \, , \; ([B \Box D_1]_{\equiv}, [D_1]_{\equiv}) \; : \; [D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(ii) $\mathcal{R} = \{ ([D_1 \Box B_2]_{\equiv}, [B_2 \Box D_1]_{\equiv}), ([B_1 \Box D_2]_{\equiv}, [D_2 \Box B_1]_{\equiv}) \; :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(iii) $\mathcal{R} = \{ ([(D_1 \Box B_2) \Box B_3]_{\equiv}, [D_1 \Box (B_2 \Box B_3)]_{\equiv}) \, ,$
$([(B_1 \Box D_2) \Box B_3]_{\equiv}, [B_1 \Box (D_2 \Box B_3)]_{\equiv}), ([(B_1 \Box B_2) \Box D_3]_{\equiv}, [B_1 \Box (B_2 \Box D_3)]_{\equiv}) :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv}), [D_3]_{\equiv} \in Reach([\overline{B_3}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(iv) $\mathcal{R} = \{ ([D_1; (B_2 \Box B_3)]_{\equiv}, [(D_1; B_2) \Box (B_1; B_3)]_{\equiv}) \, ,$
$([D_1; (B_2 \Box B_3)]_{\equiv}, [(B_1; B_2) \Box (D_1; B_3)]_{\equiv}),$
$([B_1; (D_2 \Box B_3)]_{\equiv}, [(B_1; D_2) \Box (B_1; B_3)]_{\equiv}),$
$([B_1; (B_2 \Box D_3)]_{\equiv}, [(B_1; B_2) \Box (B_1; D_3)]_{\equiv}) \; :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv}), [D_3]_{\equiv} \in Reach([\overline{B_3}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(v) $\mathcal{R} = \{ ([(D_1 \Box B_2); B_3]_{\equiv}, [(D_1; B_3) \Box (B_2; B_3)]_{\equiv}) \, ,$
$([(B_1 \Box D_2); B_3]_{\equiv}, [(B_1; B_3) \Box (D_2; B_3)]_{\equiv}),$
$([(B_1 \Box B_2); D_3]_{\equiv}, [(B_1; D_3) \Box (B_2; B_3)]_{\equiv}) \, ,$
$([(B_1 \Box B_2); D_3]_{\equiv}, [(B_1; B_3) \Box (B_2; D_3)]_{\equiv}) \; :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv}), [D_3]_{\equiv} \in Reach([\overline{B_3}]_{\equiv}) \; \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(vi) $\mathcal{R} = \{\ ([D_1 \| D_2]_{\equiv}, [D_2 \| D_1]_{\equiv})\ :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv})\ \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

(vii) $\mathcal{R} = \{\ ([(D_1 \| D_2) \| D_3]_{\equiv}, [D_1 \| (D_2 \| D_3)]_{\equiv})\ :$
$[D_1]_{\equiv} \in Reach([\overline{B_1}]_{\equiv}), [D_2]_{\equiv} \in Reach([\overline{B_2}]_{\equiv}), [D_3]_{\equiv} \in Reach([\overline{B_3}]_{\equiv})\ \} \cup \{([\underline{fail}]_{\equiv}, [\underline{fail}]_{\equiv})\}$

The bisimulation relation has been introduced between classes of dynamic terms, but we can extend it to choreographies, as follows:

Definition 10. We say that two coreographies (A_1, A_2) and (A'_1, A'_2) are bisimilar if and only if $A_1 \sim A'_1$ and $A_2 \sim A'_2$ (taking these activities with the extended syntax).

5. Case study

This case study concerns a typical Internet purchase process, which consists of four participants: the customer, the provider and two deliverers, A and B, in which we introduce some specific time restrictions. The main time restriction is due to the type of the product delivered, i.e., we classify products into two types, perishable or imperishable. Any perishable product has a time restriction due to its use-by date or expiry date (both terms are used as "expiration"). This restriction is the determining factor in deciding which deliverer will transport the product to the customer, because each has different delivery days. Another restriction, in this system, is the delivery area of each deliverer. A deliverer's area has approximately a 50 km radius and B deliverer's area is about 125 km.

Therefore, when a customer purchases a product the system must consider these two kinds of restriction: time and spatial. The product expiration and the delivery day of each deliver are the time constraints and the client's distance and radius of each deliverer are the spatial restrictions. Thus, the purchase process consists of three options. Option (a) the client's distance is inside the area covered by A and the product is perishable with an expiry period greater than A delivery day, then A will be the deliverer. Option (b) in this case the restrictions are fulfilled by B, hence he will be the deliverer. The last case, option (c), corresponds to the case where neither deliverer can fulfill the restrictions, and thus the client must pick up the product at the store by himself.

Let us now introduce the syntax for the choreography corresponding to this example, *DE*:

$$DE ::= (A_1, A_2)$$
$$A_1 ::= D_0\ ;\ (D_1 \square D_2)$$

The *life-line* A_1 starts when the client orders some product with (D_0). This term consists of 5 basic terms where the basic information for the order is established, information such as the identification of the purchased product (215), the distance from the client to the seller (10 km) and the product type (perishable or imperishable, in this case perishable). Then:

$$D_0 ::= B_1\ ;\ B_2\ ;\ B_3;\ B_4\ ;\ B_5$$
$$B_1 ::= assign(r_{Client}, Client.ID\_Product, 215)$$
$$B_2 ::= assign(r_{Client}, Client.Distance, 10)$$
$$B_3 ::= inter(r_{Client}, r_{Seller}, Client.ID\_Product, Seller.ID\_Product, \infty)$$
$$B_4 ::= inter(r_{Client}, r_{Seller}, Client.Distance, Seller.Distance, \infty)$$
$$B_5 ::= assign(r_{Seller}, Product.Type, Perishable)$$

After the client has sent the order with D_0, we have two options, executing either D_1 or D_2. The first option, D_1, represents the case in which the product is perishable:

$$D_1 ::= workunit(g_1, true, false, D_{w1})$$
$$where\ g_1 ::= (Product.Type = Perishable)$$
$$D_{w1} ::= D_{10}\ ;\ (D_{11} \square D_{12} \square D_{13})$$

D_{10} corresponds to the assignments and interactions that the different roles execute; specifically we have established that the expiration for the product is 20 days, the A deliverer takes 7 days to attend his requests, and has a radius of 50 km, whereas B deliverer takes 15 days for his requests, with a radius of 125 km. The syntax of D_{10} is as follows:

$$D_{10} ::= B_{100} \parallel (B_{101}\ ;\ B_{102}\ ;\ B_{103}\ ;\ B_{104}) \parallel (B_{105}\ ;\ B_{106}\ ;\ B_{107}\ ;\ B_{108})$$
$$B_{100} ::= assign(r_{Seller}, Product.expiry, clock + 20)$$
$$B_{101} ::= assign(r_{DLA}, DLA.Delivery\_A, clock + 7)$$
$$B_{102} ::= inter(r_{DLA}, r_{seller}, DLA.Delivery\_A, Seller.Delivery\_A, 0)$$
$$B_{103} ::= assign(r_{DLA}, DLA.Radius\_1, 50)$$

$B_{104} ::= inter(r_{DLA}, r_{seller}, DLA.Radius\_1, Seller.Radius\_1, 0)$
$B_{105} ::= assign(r_{DLB}, DLB.Delivery\_B, clock + 15)$
$B_{106} ::= inter(r_{DLB}, r_{seller}, DLB.Delivery\_B, Seller.Delivery\_B, 0)$
$B_{107} ::= assign(r_{DLA}, DLA.Radius\_2, 125)$
$B_{108} ::= inter(r_{DLA}, r_{seller}, DLB.Radius\_1, Seller.Radius\_2, 0)$

After the execution of D_{10}, the seller has all the information required to make a decision, between three different options: D_{11}, D_{12} or D_{13}.

- D_{11}: corresponds to the case in which the product is delivered by A:

$D_{11} ::= workunit(g_{11}, true, false, D_{w11})$
$where\ g_{11} ::= ((Product.Expiry > Seller.Delivery\_A) \wedge (Seller.Distance < Seller.Radius_1))$
$D_{w11} ::= B_{111}\ ;\ B_{112}$
$B_{111} ::= inter(r_{seller}, r_{DLA}, Seller.ID\_Product, DLA.ID\_Product, Seller.Delivery\_A - clock)$
$B_{112} ::= inter(r_{DLA}, r_{client}, DLA.Delivery\_A, Client.Delivery\_Day, Product.Expiry - clock)$

  B_{111} represents the interaction between the seller and the A deliverer. The time-out for this interaction is established in order to consider the A delivery time restrictions. B_{112} defines an interaction between the A deliverer and the client, in order to inform the client about the delivery day.

- D_{12}: represents the case in which the product is delivered by B:

$D_{12} ::= workunit(g_{12}, true, false, D_{w12})$
$where\ g_{12} ::= ((Product.Expiry > DLB.Delivery\_Day) \wedge (Client.Distance < DLB.Radius))$
$D_{w12} ::= B_{121}\ ;\ B_{122}$
$B_{121} ::= inter(r_{seller}, r_{DLB}, Seller.ID\_Product, DLB.ID\_Product, Seller.Delivery\_B - clock)$
$B_{122} ::= inter(r_{DLB}, r_{client}, DLB.Delivery\_B, Client.Delivery_Day, Product.Expiry - clock)$

- D_{13}: represents the case in which both time and spatial restrictions cannot be fulfilled, so the client is informed that he must personally pick up the product at the store.

$D_{13} ::= workunit(g_{13}, true, false, D_{13})$
$where\ g_{13} ::= ((\neg g_{11}) \wedge (\neg g_{12}))$
$D_{13} ::= B_{131}\ ;\ B_{132}$
$B_{131} ::= assign(r_{seller}, Seller.Pick\_Up\_At\_Store, true)$
$B_{132} ::= inter(r_{seller}, r_{client}, Seller.Pick\_Up\_At\_Store, Client.Pick\_Up\_At\_Store, Product.Expiry - clock)$

Finally, the specification of D_2 follows, which corresponds to an imperishable product:

$D_2 ::= workunit(g_2, true, false, D_{w2})\ \ where\ g_2 ::= Product.Type \neq Perishable$
$D_{w2} ::= D_{20}\ ;\ (D_{21} \square D_{22} \square D_{23})$
$D_{20} ::= (B_{201}\ ;\ B_{202}\ ;\ B_{203}\ ;\ B_{204}) \parallel (B_{205}\ ;\ B_{206}\ ;\ B_{207}\ ;\ B_{208})$
$B_{201} ::= assign(r_{DLA}, DLA.Delivery\_A, 7 + clock)$
$B_{202} ::= inter(r_{DLA}, r_{seller}, DLA.Delivery\_A, Seller.Delivery\_A, 0)$
$B_{203} ::= assign(r_{DLA}, DLA.Radius\_1, 50)$
$B_{204} ::= inter(r_{DLA}, r_{seller}, DLA.Radius\_1, Seller.Radius\_1, 0)$
$B_{205} ::= assign(r_{DLB}, DLB.Delivery\_B, 15 + clock)$
$B_{206} ::= inter(r_{DLB}, r_{seller}, DLB.Delivery\_B, Seller.Delivery\_B, 0)$
$B_{207} ::= assign(r_{DLA}, DLA.Radius\_2, 125)$
$B_{208} ::= inter(r_{DLA}, r_{seller}, DLB.Radius\_1, Seller.Radius\_2, 0)$
$D_{21} ::= workunit(g_{21}, true, false, D_{w21})$
$where\ g_{21} ::= (Seller.Distance < DLA.Radius)$
$D_{w21} ::= B_{211}\ ;\ B_{212}$

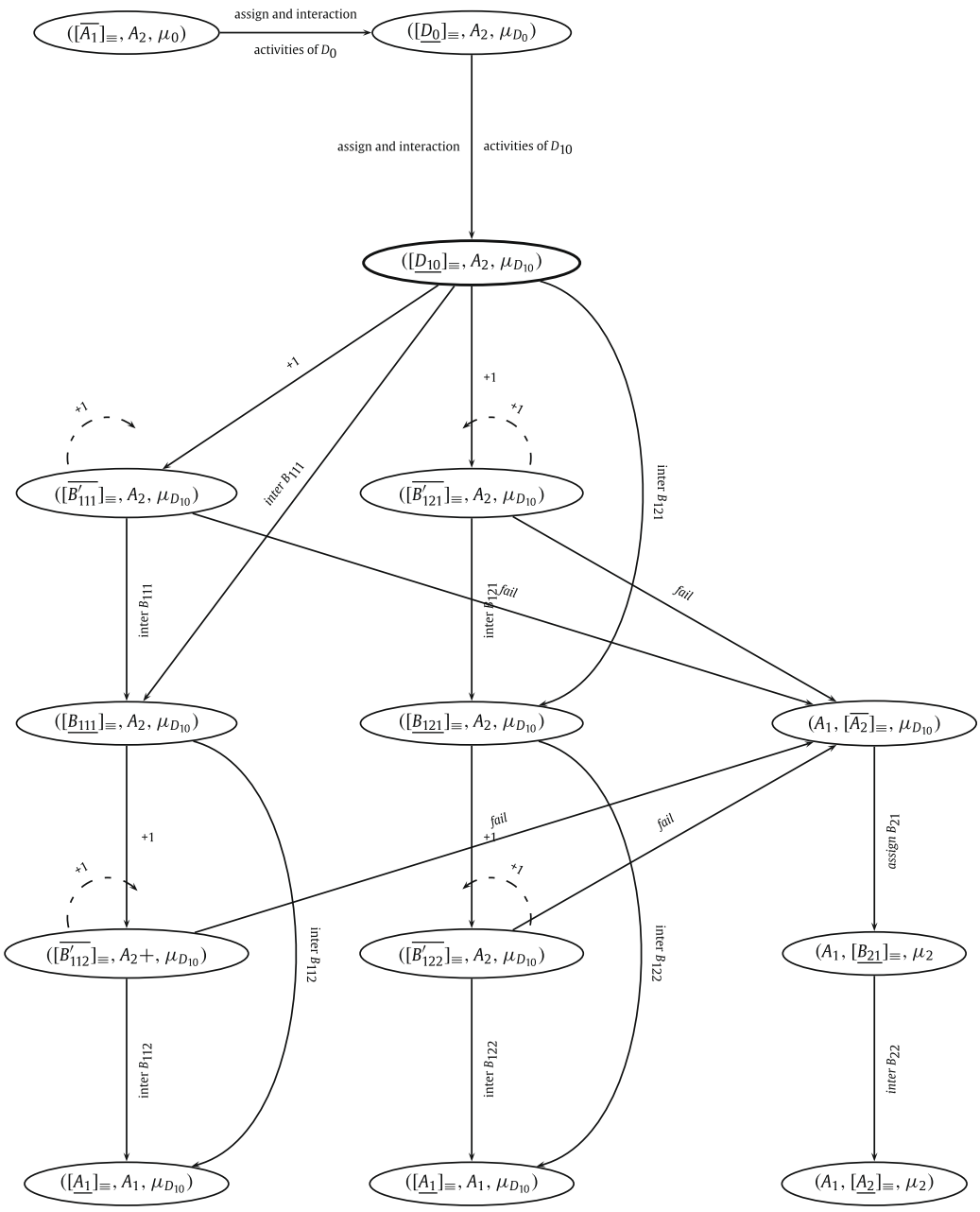

Fig. 4. A piece of the labelled transition system for the deliverers example.

$B_{211} ::= inter(r_{seller}, r_{DLA}, Seller.ID\_Product, DLA.ID\_Product, Seller.Delivery\_A - clock)$
$B_{212} ::= inter(r_{DLA}, r_{client}, DLA.Delivery_1, Client.Delivery_Day, Seller.Delivery\_A - clock)$
$D_{22} ::= workunit(g_{22}, true, false, D_{w22})$
where $g_{22} ::= (Client.Distance < DLB.Radius)$
$D_{w22} ::= B_{221} \; ; \; B_{222}$
$B_{221} ::= inter(r_{seller}, r_{DLB}, Seller.ID\_Product, DLB.ID\_Product, Seller.Delivery\_B - clock)$
$B_{222} ::= inter(r_{DLB}, r_{client}, DLB.Delivery_2, Client.Delivery_Day, Seller.Delivery\_B - clock)$
$D_{23} ::= workunit(g_{23}, true, false, D_{23})$

$$\begin{aligned}
&\text{where } g_{23} ::= ((\neg g_{21}) \wedge (\neg g_{22}))\\
&D_{23} ::= B_{131} \ ; \ B_{132}\\
&B_{231} ::= assign(r_{seller}, Seller.Pick_Up_At_Store, true)\\
&B_{232} ::= inter(r_{seller}, r_{client}, Seller.Pick_Up_At_Store,\\
&\qquad\qquad Client.Pick_Up_At_Store, \infty)
\end{aligned}$$

In the case of error, the exception block is raised (A_2), which consists of a sequence of two basic activities that inform the client about it:

$$\begin{aligned}
&A_2 ::= B_{21} \ ; B_{22}\\
&B_{21} ::= assign(r_{seller}, Seller.Error, true)\\
&B_{22} ::= inter(r_{seller}, r_{client}, Seller.Error, Client.Error, \infty)
\end{aligned}$$

Fig. 4 shows a part of the labelled transition system of this choreography, where we have simplified the figure in order to represent the main transitions in the system. The dashed arcs labelled with $+1$ are used in the figure to indicate the possibility of time elapsing up to a certain amount of time, due to the presence of a time-out.

6. Conclusions

In this paper we have presented a barred operational semantics for a relevant subset of WS-CDL, in which timing aspects of composite Web Services have been considered. The official semantics of WS-CDL [22] is defined in a textual manner. Thus, an important advantage of the provided semantics is that it can be used as alternative to the textual document with the purpose of obtaining the WS-CDL semantics in a more rigorous way. In fact, as we have seen in this paper, some points of the WS-CDL semantics are not completely described, and a formalization serves to detect these gaps too.

We have defined for this purpose a meta-model of WS-CDL, capturing its more relevant constructions. This meta-model assumes a single choreography, instead of a hierarchy of choreographies. This assumption has been introduced for simplicity, but it would not be problematic to expand the model to include this additional feature, as well as the *perform* activity, which is used to invoke a choreography from another one. Finalizer blocks have not also been considered in the meta-model, since they require a hierarchy of choreographies. These could also be included in an extension of the model, as well as the corresponding *finalize* activities, which are used to activate the finalizer blocks of some performed choreographies. For the choreography description the proposed meta-model uses the main WS-CDL elements, it divides the choreography into two parts, its life-line and its exception block, and in both parts the main activity structures are used.

As has been seen in the introduction, there are some related works that define other formal semantics for different languages of description of composite Web Services, but none of these works consider time as a central topic. Thus, one of the main contributions of this paper is the definition of a rigorous semantics of a subset of WS-CDL in which time has been considered both in interactions (time-outs) and in workunits, to delay the execution. We have considered the main activities of WS-CDL, including both the basic and the ordering structures, and we have defined a formal syntax for them, providing a set of operators that constitute the metamodel for which the barred operational semantics is defined. Another contribution of this paper is that this operational semantics is defined by using *barred* terms, which are syntactical terms that are either barred or underbarred in order to capture the current state of the described system. A relevant benefit of this barred semantics is that we do not need to split the *workunit* construction into two or more separate operators, which would be required in order to define a classical operational semantics.

We have also presented a case study in order to illustrate that timing aspects can be crucial in composite Web Services. This is a typical B2B example for which we have shown first the WS-CDL description by using our metamodel, and secondly, a piece of its labelled transition system.

The barred operational semantics has been defined for a subset of WS-CDL, which, as future work, we plan to extend allowing it to support a richer subset of WS-CDL.

References

[1] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services, Springer-Verlag, 2002.
[2] T. Andrews et al., BPEL4WS – Business Process Execution Language for Web Services. Version 1.1. May 2003. <http://www.ibm.com/developerworks/library/specification/ws-bpel/>.
[3] J.C.M. Baeten, C.A. Middelburg, EATCS Monographs Series, Springer-Verlag, 2002.
[4] E. Best, R. Devillers, M. Koutny, EATCS, Springer, 2001.
[5] M. Bravetti, G. Zavattaro, Service oriented computing from a process algebraic perspective, J. Logic Algebraic Program. 70 (1) (2007) 3–14.
[6] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo, Formalizing web service choreography, in: WS-FM'04. Electronic Notes in Theoretical Computer Science, vol. 105, 2004, pp. 73–94.
[7] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, Choreography and orchestration: a synergic approach for system design, Third International Conference of Service Oriented Computing, Lecture Notes in Computer Science, vol. 3826, 2005, pp. 228–240.
[8] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, Choreography and orchestration conformance for system design, in: Eighth Int. Conference on Coordination Models and Languages. Lecture Notes in Computer Science, vol. 4038, 2006, pp. 63–81.
[9] M. Carbone, K. Honda, N. Yoshida, Calculus of global interaction based on session types, Electron. Notes Theoret. Comput. Sci. 171 (3) (2007) 127–151.

[10] M. Carbone, K. Honda, N. Yoshida, A theoretical basis of communication-centred concurrent programming, Electron. Notes Theoret. Comput. Sci. 209 (2008) 125–133.
[11] G. Díaz, J.J. Pardo, M.E. Cambronero, V. Valero, F. Cuartero, Automatic translation of WS-CDL choreographies to timed automata, in: Proceedings Second International Workshop on Web Services and Formal Methods, WS-FM'05, Lecture Notes in Computer Science, vol. 3670, 2005, pp. 230–242.
[12] R. Hamadi, B. Benatallah, A Petri net-based model for web service composition, in: Proceedings of the 14th Australasian Database Conference, vol. 17, 2003, pp. 191–200.
[13] A. Lapadula, R. Pugliese, F. Tiezzi, Calculus for orchestration of web services, Lect. Notes Comput. Sci. 4421 (2007) 33–47.
[14] R. Lucchi, M. Mazzara, A pi-calculus based semantics for WS-BPEL, Web Services and Formal Methods, J. Logic Algebraic Program. 70 (1) (2007) 96–118.
[15] M. Mazzara, Timing Issues in Web Services Composition, EPEW2005 and WS-FM 2005, Lecture Notes in Computer Science, vol. 3670, 2005, pp. 287–302.
[16] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[17] R. Milner, J. Parrow, D. Walker, A Calculus of mobile processes, part I and II, Inform. Comput. 100 (1) (1992) 1–40.
[18] Z. Qiu, X. Zhao, C. Cai, H. Yang, Towards the theoretical foundation of choreography, in: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, 2007, pp. 973–982.
[19] G. Salaun, L. Bordeaux, M. Schaerf, Describing and reasoning on web services using process algebra, in: Second International Conference on Web Services, IEEE Computer Society Press, 2004.
[20] V. Valero, M.E. Cambronero, G. Díaz, H. Macià, A Petri net approach for the design and analysis of web services choreographies, J. Logic Algebraic Program., in press.
[21] World Wide Web Consortium (W3C). <http://www.w3.org/>.
[22] Web Services Choreography Description Language Version 1.0 (WS-CDL). <http://www.w3.org/TR/ws-cdl-10/>.
[23] Web Service Choreography Interface (WSCI) 1.0, 2002. <http://www.w3.org/TR/wsci>.
[24] H. Yang, X. Zhao, Z. Qiu, G. Pu, S. Wang, A formal model for web service choreography description language (WS-CDL), in: International Conference on Web Services (ICWS'06), IEEE Computer Society Press, 2006, pp. 893–894.
[25] W.L. Yeung. Mapping WS-CDL and BPEL into CSP for behavioural specification and verification of web services, in: Proceedings of 4th IEEE European Conference on Web Services 2006 (ECOWS'06), IEEE Computer Society, 2006, pp. 297–305.
[26] Q. Zongyan, Z. Xiangpeng, C. Chao, Y. Hongli, Towards the theoretical foundation of choreography, in: Proceedings of the 16th International World Wide Web Conference, ACM, 2007, pp. 973–982.